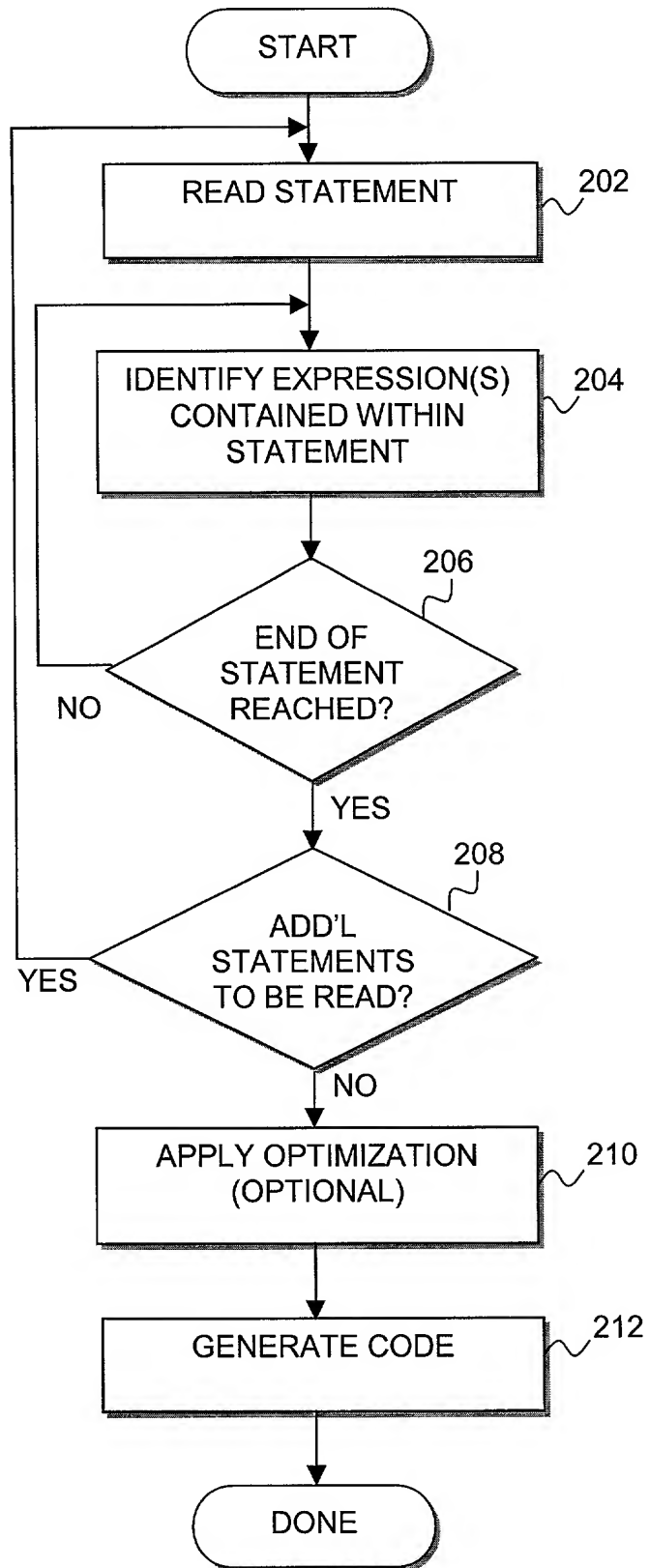


FIG. 1

**FIG. 2**

```

300      (A)      XML x;           // an XML variable
      (B)      XML y;           // another XML variable
      (C)      int i;           // an integer variable
      (D)      float f;         // a floating point variable

```

```

310      (A)      XML p = <person><name>John</name><age>25</age></person>;
      (B)      e = <employees>
      (C)          <employee id="1"><name>Joe</name><age>20</age></employee>
      (D)          <employee id="2"><name>Sue</name><age>30</age></employee>
      (E)      </employees>;

```

```

320      (A)      for (i = 0; i < 10; i++)
      (B)          e[i] = <employee id={i}>
      (C)              <name>{names[i].toUpperCase()}</name>
      (D)              <age>{ages[i]}</age>
      (E)          </employee>;

```

FIG. 3A

```

330  (A) XML John = "<employee><name>John</name><age>25</age></employee>";
      (B) XML Sue = "<employee><name>Sue</name><age>32</age></employee>";
      (C) String tagName = "employees";
      (D) XML employees = "<"+ tagName +">" + John + Sue + "</"+ tagName +">";
      (E) XML employees = <employees>{John + Sue}</employees>;

340  (A) XML e = <employee id="1"><name>John</name><age>25</age></employee>;
      (B) String name = e.employee.name;           // get the name of the employee
      (C) e.employee.name = "Fred";               // set the name of the employee
      (D) XML ref = e.employee;                    // returns a reference to the <employee> element

350  (A) int empid = e.employees.employee[1].@id;    // get the id attribute of the second employee
      (B) e.employees.employee[3].@id = 10;         // set the id attribute of the fourth employee

355  (A) e = <employees>
      <employee id="4" bossid="2"><name>Jim</name><age>25</age></employee>
      <employee id="3" bossid="2"><name>Joe</name><age>20</age></employee>
      <employee id="2" bossid="1"><name>Sue</name><age>30</age></employee>
      <employee id="1"><name>Mr. CEO</name><age>35</age></employee>
      </employees>;

      (B) bossids = e.employee.@bossid              // get all the boss ids (2, 2, 1)
      (C) bosses = e.employee.@bossid.name          // get names of all bosses (Sue, Sue, Mr. CEO)
      (D) bossages = e.employee.@bossid.age         // get ages of all bosses (30, 30, 35)

```

FIG. 3B

```

360 (A) e = <employees>
      (B)   <employee id="1"><name>Joe</name><age>20</age></employee>
      (C)   <employee id="2"><name>Sue</name><age>30</age></employee>
      (D)   </employees>
      (E)   names = e..name;           // get all the names in e

```

```

370 (A) e = <employees>
      (B)   <employee id="0"><name>Joe</name><age>20</age></employee>
      (C)   <employee id="1"><name>Sue</name><age>30</age></employee>
      (D)   </employees>
      (E)   suesAge = e.employees.employee[1].age;           // get Sue's age
      (F)   e.employees.employee[1].name = "Fred";           // Change Sue's name to Fred

```

```

380 (A) For (n in e..name)
      (B)   Print ("Employee name: " + n);           // print out all the names in e
      (C)   For (i = 0; i < e..name.length; i++)
      (D)     print("Employee name:" + e..name[i]);

```

FIG. 3C

```

385 (A) John = e..employee.(name=="John");           //employees with name John
      emp=e..employee.(@id==1).name;               //name of employee with id 1
      twoEmployees=e..employee.(@id==0 || @id==1); //employees with id's 0 and 1

      (D) i=0;
          twoEmployees = new Array(2);
          for (p in e..employee){
              if (p.@id==0 || p.@id==1){
                  twoEmployees[i++]=p;
              }
          }

```

```

390 (A) e = <employees>
      <employee id="0"><name>Jim</name><age>25</age></employee>
      <employee id="1"><name>Joe</name><age>20</age></employee>
      <employee id="2"><name>Sue</name><age>30</age></employee>
      <employee id="3"><name>Mr. CEO</name><age>35</age></employee>
      </employees>

      (B) // replace Joe with Bill using a document fragment
          e..employee[1] = <><name>Bill</name><age>25</age><hobby>Skiing</hobby></>;

      (C) // replace Jim with Ann without using document fragments
          e.employee[0].name = "Ann";
          e.employee[0].age = 32;
          e.employee[0].appendChild("hobby");
          e.employee[0].hobby = "Kayaking";

```

FIG. 3D

```

// Xpath function to retrieve an employee with a particular name from an employee list
language(XPATH) function employeeByName(e, name) {
    $e/employees/employee[name = $name]
}

// XML function to build an employee record given a name and age
language (XML) function createEmployee(name, age)
{
    <employee><name>{$name}</name><age>{$age}</age></employee>
}

// XQUERY function to get the name and average book price from a list of books
// given as a parameter. Each book contains a publisher and price.
language (XQUERY) function publisherInfo(booklist)
{
    for $p in distinct(document($booklist//publisher)
    let $a := avg($booklist//book[publisher = $p]/price)
    return
        <publisher>
            <name>{$p/text()}</name>
            <aveprice>{$a}</aveprice>
        </publisher>
}

// SQL function to get the employee with a given ID
language(SQL) function employeeByID(id)
{
    select * from e.employees.employee where id == $id;
}

```

FIG. 3E

FUNCTION	RESULT
parent()	Returns The Parent Of The Xml Value
children()	Returns All The Children Of The Xml Value
childIndex()	Returns The Ordinal Position Of The Xml Value Within Its Parent
xpath(expression)	Evaluates The Xpath Expression Using The Xml Value As The Context Node
appendChild(<i>child</i>)	Inserts A New <i>Child</i> Node After The Existing Children Of The Xml Value
prependChild(<i>child</i>)	Inserts A New <i>Child</i> Node Before The Existing Children Of The Xml Value
copy()	Returns A Complete Copy Of The Xml Value
delete(<i>childList</i>)	Deletes The Children Referenced By <i>ChildList</i>
innerHTML(<i>content</i>)	Replaces The Entire Contents Of The Xml Value With New <i>Content</i>
validate()	Validates the XML value conforms with it's associated schema type. If not, it returns errors.

FIG. 3F

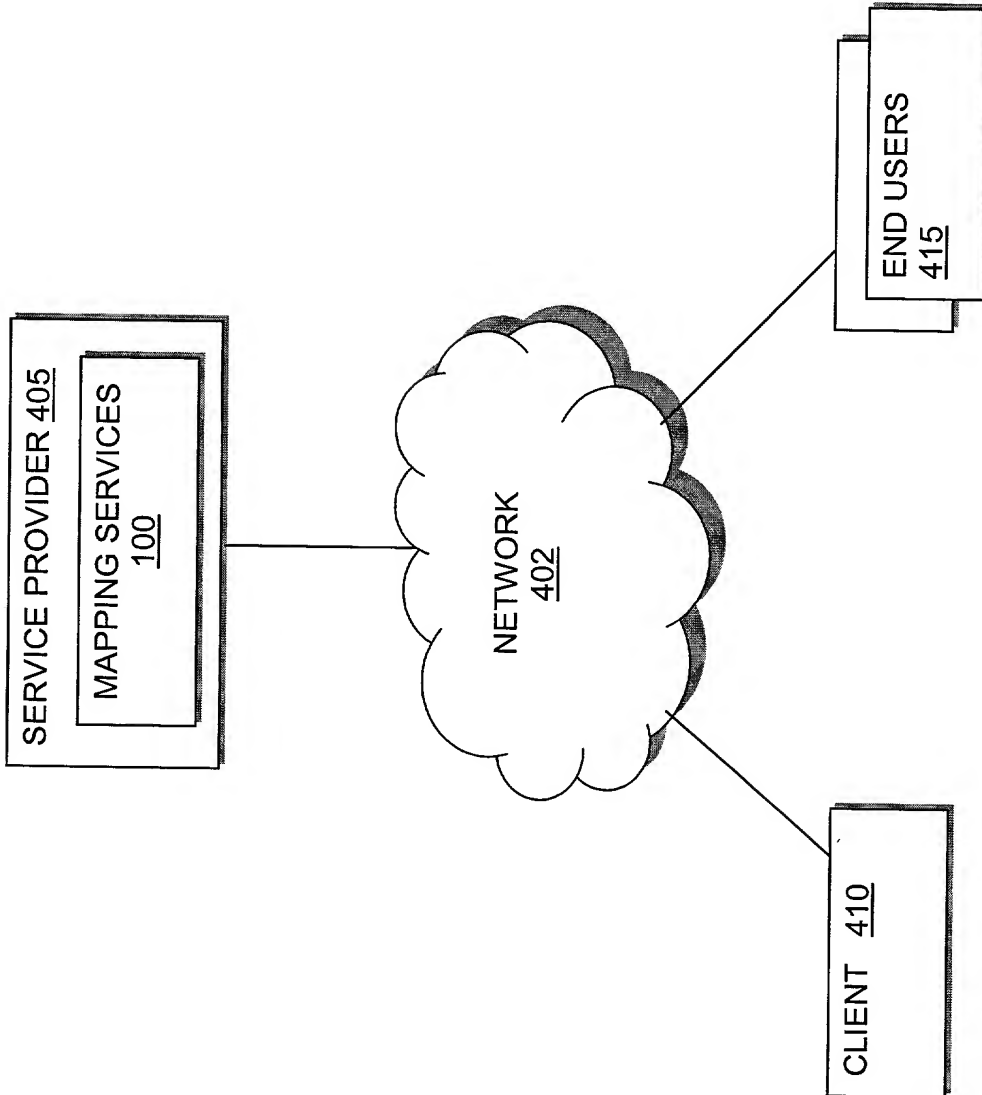


FIG. 4

```
Function FromXML(XML availableat, Point[] points) {  
  
    int i = 0;  
    points = new Point[availableat..address.length()];    // one point for each address  
  
    // for each address, compute the display label and point location  
    for (a in availableat..address) {  
  
        points[i].label = a.parent().name + '\n'  
            + a.street + '\n'  
            + a.city + ', ' + a.state + ', ' + a.zip;  
  
        // The label is 3 lines long  
        // line 1: store name  
        // line 2: street address  
        // line 3: city, state, zip  
  
        // compute map location from the address  
        points[i].location = new LatitudeLongitude(a.street, a.city, a.state, a.zip);  
        i++;  
    }  
}
```

FIG. 5A

```

Function ToXML(Point[] points, XML availableat) {
    // build the root node
    availableat = <available-at></available-at>;

    // for each point, add a store element
    for (p in points) {
        lines = p.label.split("\n");
        citystatezip = lines[2].split(", ");
        // split the label up into its 3 lines
        // split line 3 into city, state, zip

        // append a store element representing this point
        availableat.available-at.appendChild(
            <store>
                <name>{lines[0]}</name>           // line 1 is store name
                <address>
                    <street>{lines[1]}</street>    // line 2 is street addr
                    <city>{citystatezip[0]}</city>
                    <state>{citystatezip[1]}</state>
                    <zip>{citystatezip[2]}</zip>
                </address>
            </store>);
    }
}

```

FIG. 5B

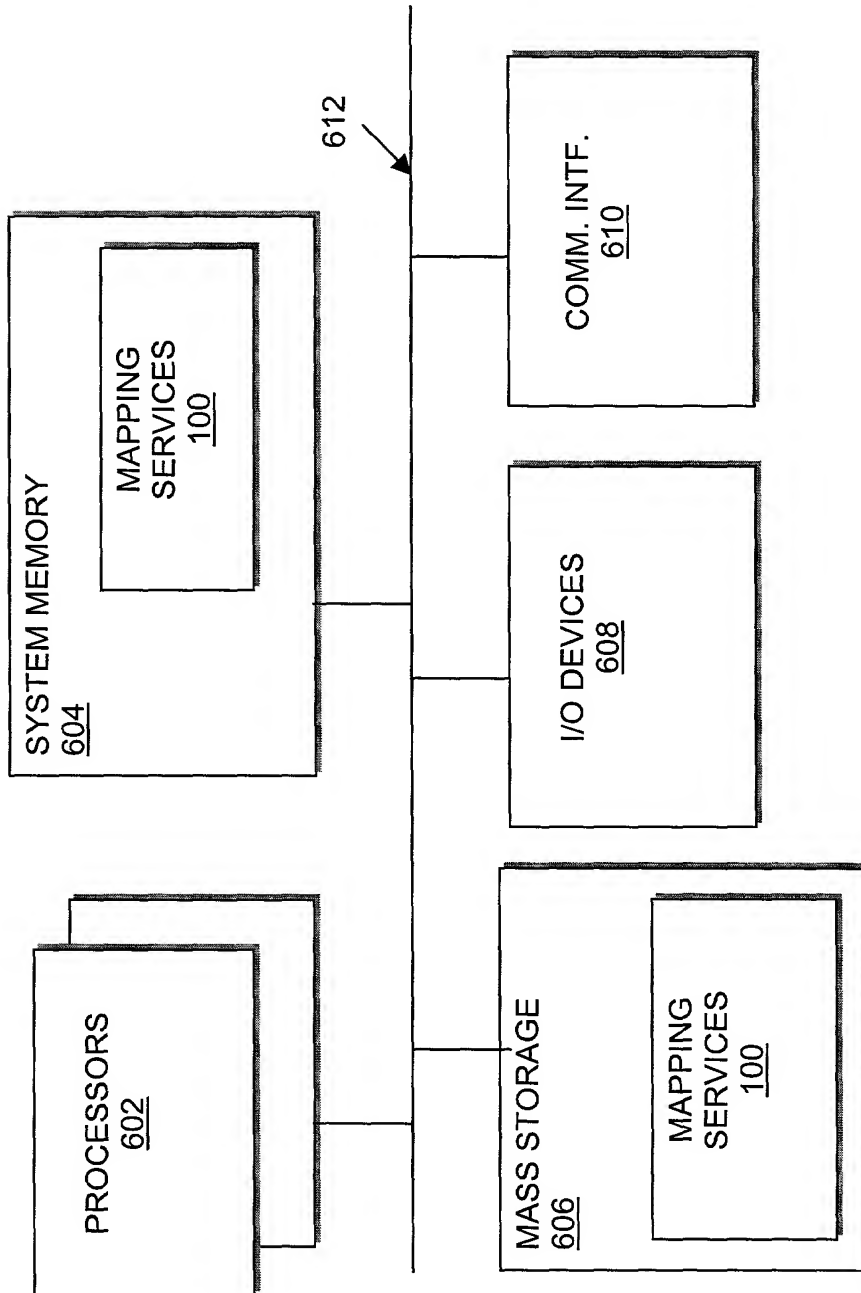


FIG. 6

```
import org.w3c.dom.*;
import org.apache.xerces.dom.*;
```

```
// class declarations omitted ...
```

```
public static void FromXML(Document availableat, Point[] points) {
    // one point for each address
    NodeList addresses = availableat.getElementsByTagName("address");
    points = new Point[addresses.getLength()];

    // for each address, compute the display label and point location
    for (int i = 0; i < addresses.getLength(); i++) {
        // get the store name, street address, city, state and zip for each address
        Element a = (Element)addresses.item(i);
        Element store = (Element)a.getParentNode();
        Node name = store.getElementsByTagName("name").item(0);
        Node street = a.getElementsByTagName("street").item(0);
        Node city = a.getElementsByTagName("city").item(0);
        Node state = a.getElementsByTagName("state").item(0);
        Node zip = a.getElementsByTagName("zip").item(0);

        // the label is three lines long
        points[i].label = name.getNodeValue() + "\n" //line 1: store name
            + street.getNodeValue() + "\n" //line 2: street address
            + city.getNodeValue() + ", " //line 3: city
            + state.getNodeValue() + ", " // state
            + zip.getNodeValue(); // zip

        // compute map location from the address
        points[i].location = new LatitudeLongitude(
            street.getNodeValue(), city.getNodeValue(),
            state.getNodeValue(), zip.getNodeValue());
    }
}
```

FIG. 7A
(PRIOR ART)

```
public static void ToXML(Point[] points, Document availableat) {
    // build the root node
    availableat = new DocumentImpl();
    Element root = availableat.createElement("available-at");

    // for each point, add a store element
    for (int i = 0; i < points.length; i++) {
        // split the label up into its 3 lines
        String name = points[i].label.substring(0,
            points[i].label.indexOf('\n'));
        String street = points[i].label.substring(name.length()+1,
            points[i].label.indexOf('\n', name.length()+1));
        String citystatezip = points[i].label.substring(
            name.length()+street.length()+2);

        // split line 3 into city, state and zip
        String city = citystatezip.substring(0,
            citystatezip.indexOf(" "));
        String state = citystatezip.substring(city.length()+2,
            citystatezip.indexOf(" ", city.length()+2));
        String zip = citystatezip.substring(
            city.length()+state.length()+4);

        // create Text Nodes for the store name, street address, city, state and zip
        Text nameText = availableat.createTextNode(name);
        Text streetText = availableat.createTextNode(street);
        Text cityText = availableat.createTextNode(city);
        Text stateText = availableat.createTextNode(state);
        Text zipText = availableat.createTextNode(zip);

        // create elements to hold the text nodes
        Element ename = availableat.createElement("name");
        Element eaddress = availableat.createElement("address");
        Element estreet = availableat.createElement("street");
        Element ecity = availableat.createElement("city");
        Element estate = availableat.createElement("state");
        Element ezip = availableat.createElement("zip");

        // attach the text nodes to their associated elements
        ename.appendChild(nameText);
        estreet.appendChild(streetText);
        ecity.appendChild(cityText);
        estate.appendChild(stateText);
        ezip.appendChild(zipText);
    }
}
```

FIG. 7B
(PRIOR ART)

```
// build the address element content
eaddress.appendChild(estreet);
eaddress.appendChild(ecity);
eaddress.appendChild(estate);
eaddress.appendChild(ezip);

// create the store element, add it's associated name and address
Element store = availableat.createElement("store");
store.appendChild(ename);
store.appendChild(eaddress);

// append a store element representing this point
root.appendChild(store);
}
```

FIG. 7C
(PRIOR ART)

```
// Use the AvailableAt Schema to enforce the rules established by product manufacturers
810 import AvailableAt.xsd;

820 Function FromXML(AvailableAt availableAt, Point[] points) {
    int i = 0;
    points = new Point[availableAt.address.length()]; // one point for each address

    // for stores that stock the product, compute the label and point from the address
830 for (a in availableAt.store.(stock-item == "yes").address) {

        // The label is 3 lines long
        points[i].label = a.parent().name + '\n' // line 1: store name
            + a.street + '\n' // line 2: street address
            + a.city + ', ' + a.state + ', ' + a.zip; // line 3: city, state, zip

        // compute map location from the address
        points[i].location = new LatitudeLongitude(a.street, a.city, a.state, a.zip);
        i++;
    }
}
```

FIG. 8A

```

840  Function ToXML(Point[] points, AvailableAt availableat) {
        // build the root node
        availableat = <available-at></available-at>;

        // for each point, add a store element
        for (p in points) {
            lines = p.label.split("\n");           // split the label up into its 3 lines
            citystatezip = lines[2].split(", ");    // split line 3 into city, state, zip

            // append a store element representing this point
            availableat.available-at.appendChild(
                <store>
                    <name>{lines[0]}</name>           // line 1 is store name
                    <stock-item>yes</stock-item>
                    <address>
                        <street>{lines[1]}</street> // line 2 is street addr
                        <city>{citystatezip[0]}</city>
                        <state>{citystatezip[1]}</state>
                        <zip>{citystatezip[2]}</zip>
                    </address>
                </store>);
        }

        // If resulting XML does not conform to the Schema, generate an error
        errors = availableat.validate();
        if (errors != null)
            throw InvalidXMLException(errors);
850

```

FIG. 8B

// Use the AvailableAt Schema to enforce the rules established by product manufacturers
import AvailableAt.xsd;

```

910 language(XQUERY) Function getStates(AvailableAt availableat){
    <states>
    FOR $state in distinct($availableat//state)
    LET $numStores := count($availableat//address[state = $state])
    RETURN <state>
        <abbreviation> $state </abbreviation>
        <stores> $numStores </stores>
    </state>
    </states>
}

```

```

Function FromXML(AvailableAt availableat, Point[] points) {
    // get the states that carry the product and # stores in each state
    XML states = getStates(availableat);
    int i = 0;
    points = new Point[states..state.length()]; // one point for each state
    // for each state, compute the appropriate label and point
    for (s in states..state) {
        // The label contains the state abbreviation and # stores
        points[i].label = s.abbreviation + '\n' + s.stores;
        // compute map location from the state
        points[i].location = new LatitudeLongitude(s.state);
        i++;
    }
}

```

FIG. 9